Caesar Cipher Method Design and Implementation Based on Java, C++, and Python Languages

Ismail M. Keshta
ismailk@dcc.kfupm.edu.sa
dr.ismail.keshta@gmail.com

***Abstract*—Today information security is a challenging factor that touches a lot of areas, including computers and communications. Message communication is kept secure through cryptography so that an eavesdropper is not able to decipher a transmitted message. One of the oldest and simplest known algorithms for cryptography is the Caesar cipher algorithm. In this paper, three programs based on Java, C++, and Python languages have been developed to implement the Caesar cipher algorithm to aid information security students and help them understand this fundamental algorithm. A code flow chart is used for each program to describe the code's flow. It also reveals the sequence of steps for the code's main methods, as well as the relationships between them. Furthermore, various technical descriptions are presented in detail for each of the methods used in both the encoding and the decoding of the messages.**

Keywords: cryptography, Caesar cipher, encryption, decryption, plain text, cipher text

I. INTRODUCTION

Desiring to securely transmit messages that cannot be understood by others is not new. In fact, people have needed to keep their communications private and secure for centuries [1]. Today digital communication systems are able to carry huge amounts of sensitive data, particularly those related to the Internet—for example, sending information about a credit card in an e-commerce transaction or using e-mail to exchange confidential trade secrets [1][2][3].

Network security is, therefore, a vital aspect of information sharing. Many technological implementations and security policies have been developed in various attempts to remove insecurities over the Internet [4][5]. The total amount of data transferred is not an important factor. What is important is how much security the channel can provide while it is transmitting data [6]. Cryptography is a technique that allows the secure transmission of data without loss of confidentiality or integrity [6][7].

The field of cryptography is vital as it deals with techniques that can convey information securely. Its aim is to allow recipients to receive their message properly while also stopping any eavesdroppers from deciphering and understanding what is written [8]. The original form of the message is called plaintext. The transmitter uses a secure system to encrypt the plaintext to hide the true meaning. This is a reversible mathematical process that produces an encrypted output, which is called ciphertext, and the algorithm used to encrypt the message is called a cipher. The science of breaking ciphers is called cryptanalysis, and a cryptanalyst tries to break the security of cryptographic systems [8][9].

It is worth noting that a ciphertext can be openly transmitted across the communication channel. However, because of the encrypted nature of this message, eavesdroppers who have access to the ciphertext should not be able to uncover its meaning, as only the intended recipient can decrypt it to recover the original plaintext for interpretation [10]. All these processes are shown in Figure 1.

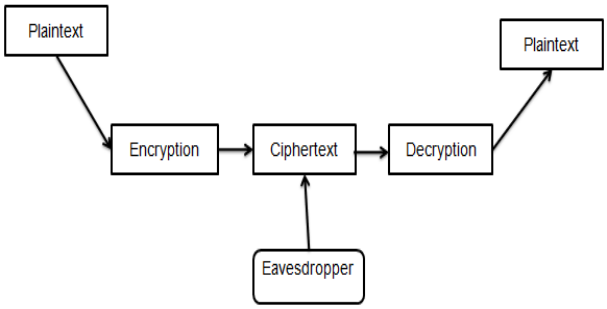

Figure 1: Block diagram of a cryptographic system

Cryptography can be split into two main types based on key distribution: symmetric key cryptography and asymmetric key cryptography. In symmetric key cryptography, the same key is used for both encryption and decryption (see Figure 2). On the other hand, asymmetric key cryptography uses different keys for encryption and decryption (see Figure 3). The two keys are related to each other mathematically, and obtaining one from the other is very hard [11].

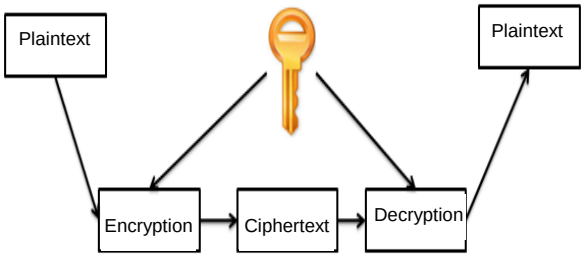

Figure 2: Symmetric key encryption

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

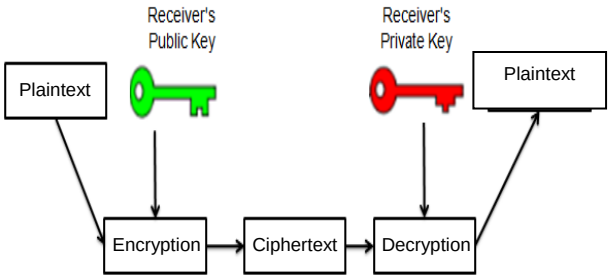

Figure 3: Asymmetric key encryption

Cryptography is a collection of various techniques called "ciphers" that are used to make things more difficult to both read and understand [11][12]. Cryptography has many uses, including Internet security and spying! There is a constant struggle between those who create ciphers and those who set out to break them.

The Caesar cipher is one of the oldest and simplest ciphers. It is no longer used in encryption systems, but strong modern ciphers use it in combination with various other operations [13][14][15]. This results in an algorithm that is more secure than its components. We describe this cipher in further detail in the following section.

II. CAESAR CIPHER

The Caesar cipher is a simple substitution cipher of historical interest. It gets its name from Julius Caesar, who used this code to send his secret messages [16][17]. Caesar encrypted his messages by moving every letter in the plaintext three positions to the right in the alphabet [18]. This cipher is based on shifted alphabets. Therefore, it is also known as a shift cipher, Caesar shift, or Caesar cipher and is used as a substitution method to evolve the encrypted text [19]. The following illustrates Caesar's method:

Plaintext:	the exam will start at eight in the morning
Ciphertext:	wkh hadp zloo vwduw dw hljkw lq wkh pruqlqj

The Caesar cipher encryption process is performed by using the following function [20] [21]:

$$E(x) = (x + n)\ mode\ 26$$

The decryption process reverses the encryption process by using the following function [22]:

$$D(x) = (x - n)\ mode\ 26$$

It is worth mentioning that the Caesar cipher method did not lasting long because of its simplicity and obvious lack of communication security [13]. Breaking this cipher is not difficult as the plaintext's own statistical information is already contained in the ciphertext. By performing frequency analysis, it is easy to reveal that the Caesar cipher has been used [1] as well as to quickly uncover the message. The amount of the shift K = 3 is defined to be the key for the Caesar cipher. Shifts by other amounts can also be used. However, there are just 25 possible shifts in the alphabet, and all possible combinations can be made quickly [1][13][20].The fact that a shift in letters is used means that it is not difficult to guess the key because it is a single private key with a space for 1 to 25 different combinations. A long cipher text can be difficult to break manually, and you could have no clue what key has been used, but it has no standing in the modern age of computers anymore as it can be easily broken through a brute force attack because only 25 possible key options available (from K=1 to K=25) [14].

III. IMPLEMENTATION OF THE CAESAR CIPHER METHOD

Three programs based on Java, C++, and Python languages have been developed to implement the Caesar cipher method using the user-defined key for the shift. Each program starts by asking the user for the source file, the target file, and the key. In particular, it asks the user to enter the file's name or the path of the file user that is needed for decoding after the decode file path/name is obtained. It then asks for the file name or path required to store the decoding of the plaintext. After both resource files are obtained, the program displays the menu to encode, decode, and copy the text file's data, according to the user's needs. After the user selection is obtained, the following can be done:

- Encode
- Decode
- Copy the data

In the encoding phase, the program rearranges the plaintext using the key defined by the user. In the decoding phase, the program converts the cipher text into plaintext using the key defined by the user. In copying the data, the program copies the exact plaintext or cipher text from an input file to the target file or output file.

The program has custom methods for each task. For the encoding phase, the class has two methods:

- ***encode*** – this method reads the input file line by line, parses each line into words, encrypts each word character by character, and writes to the output file.
- ***encodeHelper*** – this method performs a subtask for the encode method: encrypt a given character according to the key specified by the user.

For the decoding operation, the class has two methods similar to those in the encoding phase:

- ***decode*** – similar to the *encode* method, it reads the input file line by line, parses each line into words, decrypts each word character by character, and writes to the output file.
- ***decodeHelper*** – this method performs a subtask for the decode method: decrypt a given character according to the key specified by the user.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

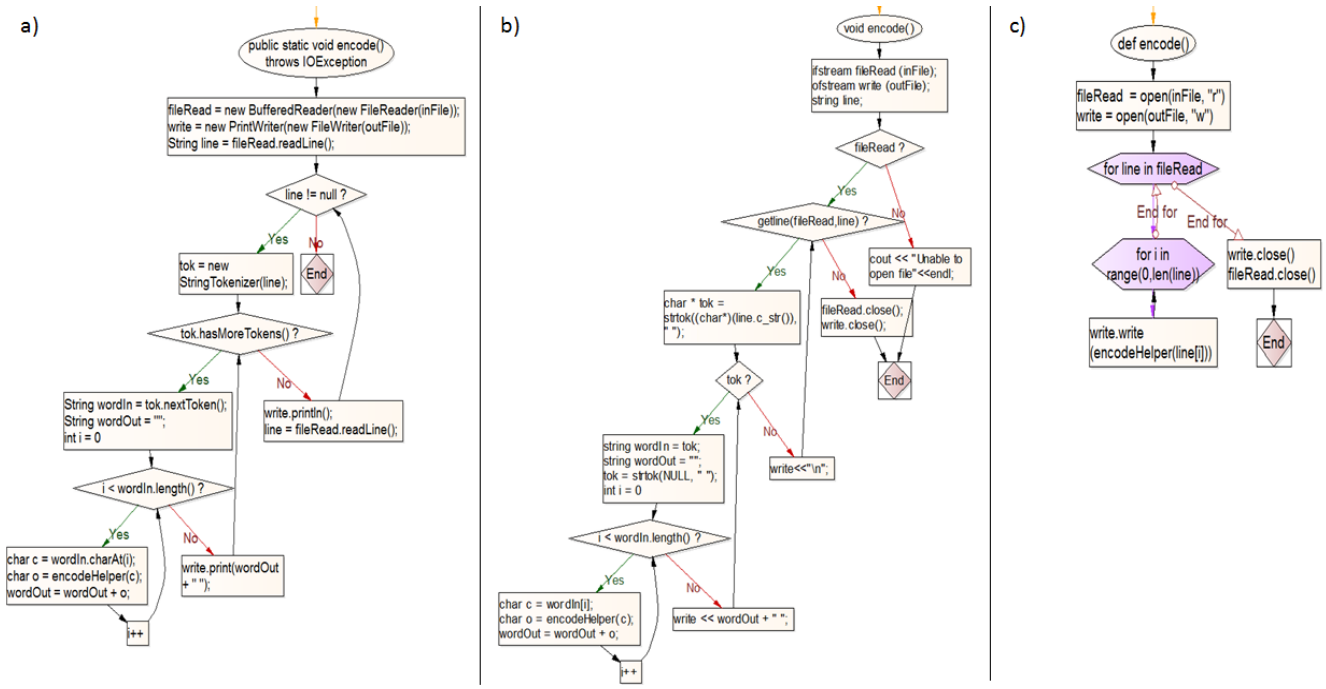

Figure 4: Code flow chart for the encode method: (a) Java, (b) C++, (c) Python

This program also has an additional method for copying one file to another, called *copy*. It copies one file to another line by line. In the following subsections, we present detailed technical descriptions for each method used in the encoding and decoding processes.

A. *encode Method*

This method allows for the encoding of the plaintext of the input file into the cipher text with respect to the key and with the aid of an encode helper. After the arrangement of the file read and write, the program starts reading the file text using a buffer reader line by line and storing the line in a string-type variable. We then split a string to read it word by word. After the line is split into words, the program starts to read the word character by character. After this, we get the output character with respect to the key and input character with the help of the encodeHelper method. It is important to note that the encode method performs the following steps:

1. Read a line.
2. If the line is null (i.e., end of the file), stop.
3. Break the lines into tokens by breaking around the space.
4. For each token of the line, perform the following:
   a) Create an empty string to store the encoded word.
   b) Get a character from the word.
   c) Encode using the encodeHelper method.
   d) Append to the encoded word created in step 4(a).
   e) Repeat steps 4(b) to 4(d) for each character in the word.
5. Write the encoded word to file.
6. Repeat steps 1 to 5 for each line of the file.

To illustrate this method, assume that we have a file that contains these two lines: "Hello there. Welcome to Caesar cipher." The method reads the file line by line; thus, line 1 is read first ("Hello there"). Then the method breaks this into tokens around a space. The two tokens are "Hello" and "there." Then the method encodes these tokens one by one. "Hello" is encoded first. The loop encrypts the word one by one. So "Hello" is encrypted to, let us say, "Fcjjm." This word is then written to file. Similarly, "There" is encoded and written to file. Then the program moves to read the next line and encode and write to file in the same way as in the previous line Figure 4 illustrates the code flow chart for this method [1].

B. *encodeHelper Method*

The method is basically used by the encode method. It returns the encoded character after a shift to the right. The shift is done by "key" characters specified at the start of the program. The only characters supported are *a*–*z* and *A*–*Z*. The rest of the characters are returned as they are and not encoded. The following steps describe how the method works:

1. Check whether the character is *a*–*z* or *A*–*Z*.
2. If not, return the character as it is.
3. Find the new character position after shifting to the right.
4. If the character after the shifting goes out of range (becomes greater than *z*), then wrap around to start.
5. If the character is within range, return that character after the shifting.

[1]All Code flow charts are available on the following link:
https://sites.google.com/site/caesarciphermethoddesign/

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

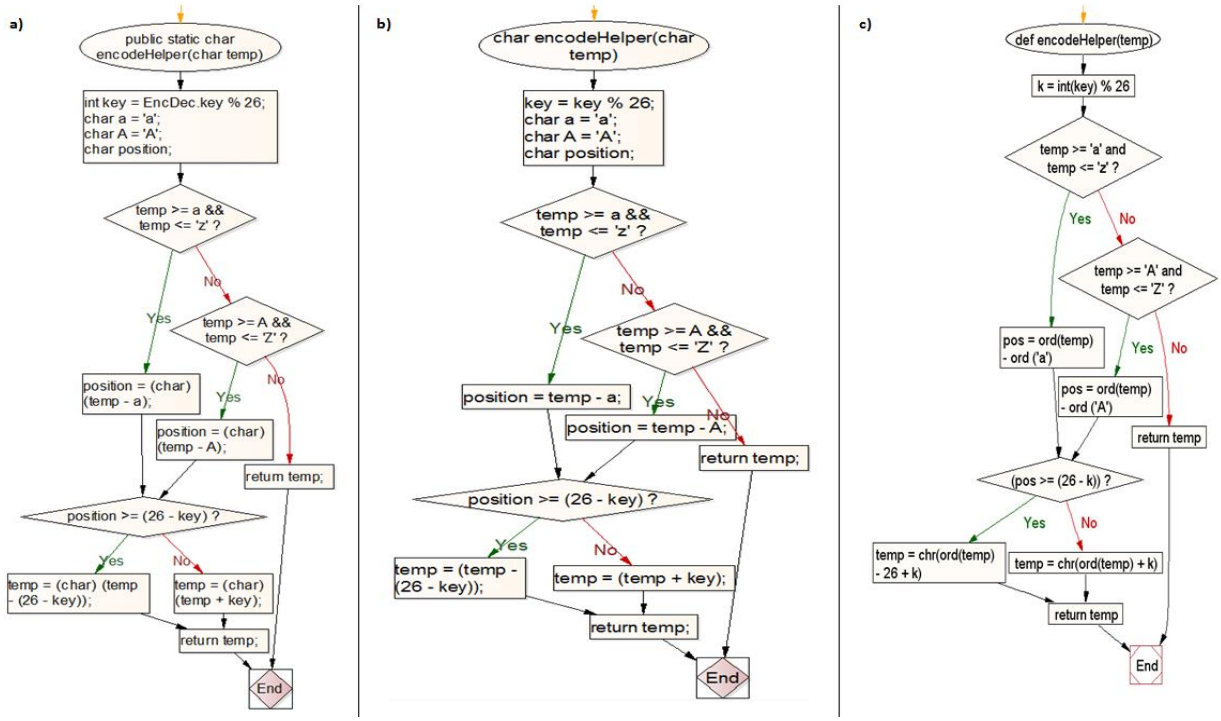

Figure 5: Code flow chart for the encodeHelper method: (a) Java, (b) C++, (c) Python

For example, let us assume that the value of key = 2 and the character to encode = c. First, we check whether the character is a–z or A–Z, which is true. Then we find the position of the character from the start of its character set. So the position of c = 2 because it is two characters away from a. Then we check whether the position + key is greater than or equal to 26 (size of the character set). In our case, 2 + 2 < 26; thus, we shift c two characters ahead, so the character that is encoded is e.

Now let us assume that the value of key = 2 and the character to encode = z. Therefore, z is a supported character that can be encoded. Now the position of z = 25. Then we check the position + key = 25 + 2 = 27 > 26. So we wrap around by subtracting 26 from the sum = 1. Thus, we shift by moving one character ahead of a. So z is encoded as b. Figure 5 describes the code flow chart for the encodeHelper method.

C. decode Method

This method first creates a buffered reader to read from a file and a print to write to the output file. Then the method reads from the file line by line. It is necessary to highlight that the *decode* method performs the following steps:

1. Read a line.
2. If the line is null (i.e., end of the file), stop.
3. Break the lines into tokens by breaking around the space.
4. For each token of the line, perform the following:
   a) Create an empty string to store the decoded word.
   b) Get a character from the word.
   c) Encode using the *decodeHelper* method.
   d) Append to the decoded word created in step 4(a).
   e) Repeat steps 4(b) to 4(d) for each character in the word.
5. Write the decoded word to file.
6. Repeat steps 1 to 5 for each line of the file.

To describe this method, consider that we have a file containing the following two lines of the decrypted text:

Fcjjmrfcpc
UcjamkcrmAycqcpAgnfcp

The method reads this line by line; thus, line 1 is read first ("*Fcjjmrfcpc*"). Then the method breaks this into tokens around a space. So the two tokens are "*Fcjjm*" and "*rfcpc.*" Then the method encodes these tokens one by one. So "*Fcjjm*" is decoded first. The loop decrypts the word one by one. So "*Fcjjm*" is decrypted to, let us say, "Hello." Then this word is written to file. Similarly, "*rfcpc*" is decoded and written to file. Then the program moves to read the next line and decode and write to file in the same way as in the previous line. It is worthwhile to note that a description of the code flow chart for the decode method is presented in Figure 6.

D. decodeHelper Method

This method returns the decoded character after a shift to the left. The shift is done by the "key" characters specified at the start of the program. The only characters supported are *a–z* and *A–Z*. The rest of the characters are returned as they are and not decoded. The following important steps show how the method works:

1. Check whether the character is a–z or A–Z.
2. If not, return the character as it is.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

3. Find the new character position after shifting to the left.
4. If the character goes out of range (becomes less than a or A), wrap around to the end of the character set.
5. If the character is within range, return that character after shifting to the left.

For example, let us assume that the value of key = –2 and the character to decode = c. First, we check whether the character is a–z or A–Z, which is true. Then we find the position of the character from the start of its character set. So the position of c = 2 because it is two characters away from a. Then we check whether the position + key is less than zero (goes left to a or A). In our case, 2 – 2 = 0, which is not less than zero, so we shift c two characters back, so the character decoded is a. For a more detailed description, see Figure 7 .

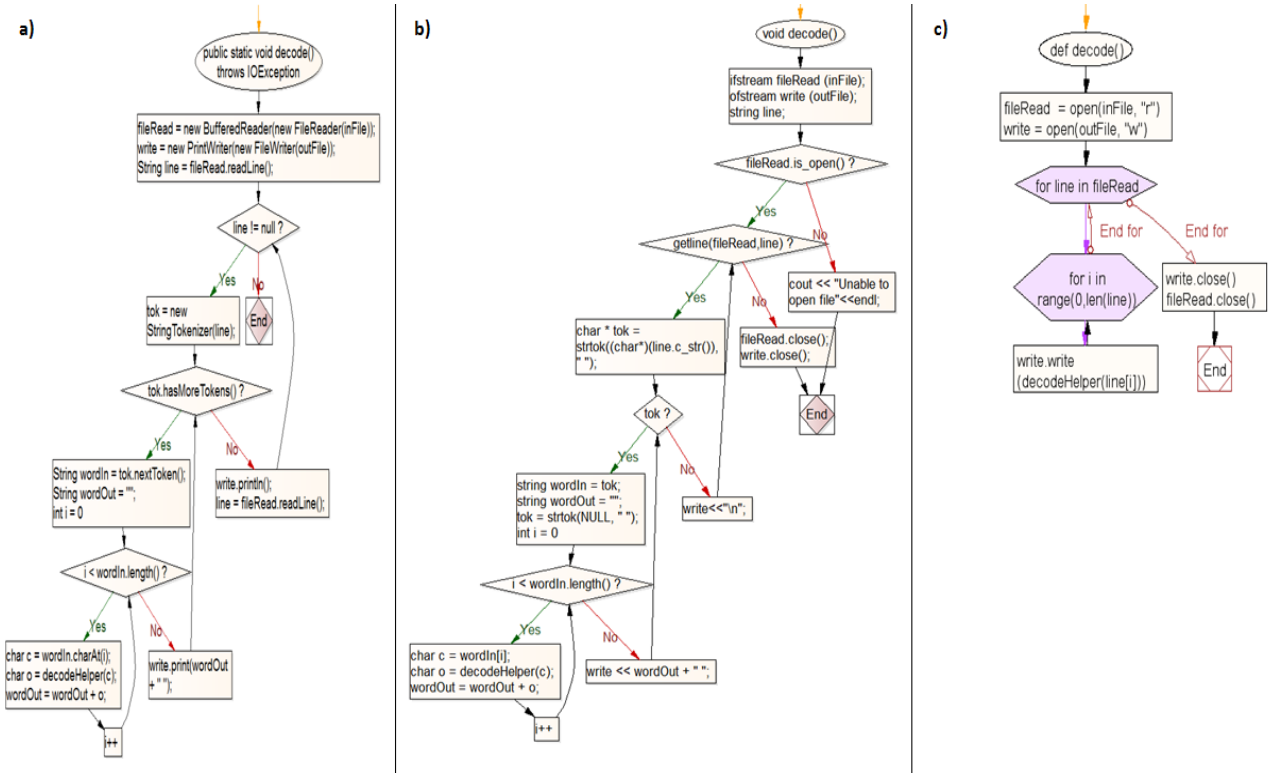

Figure 6: Code flow chart for the decode method: (a) Java, (b) C++, (c) Python

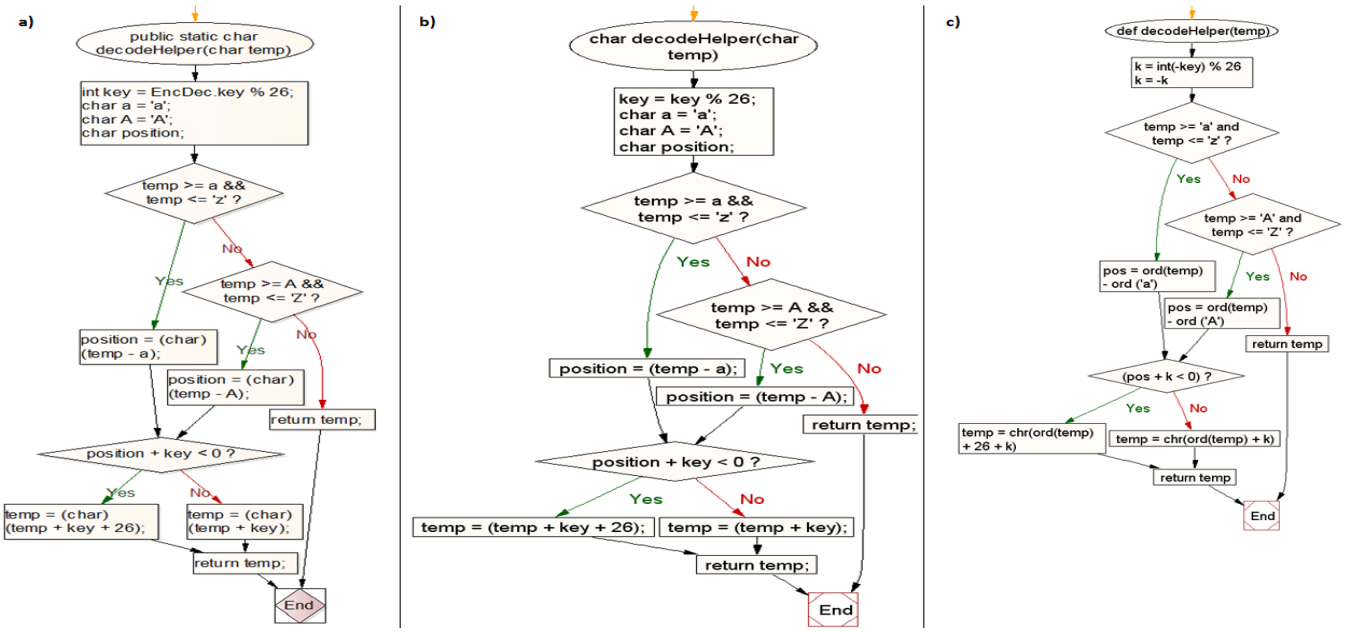

Figure 7: Code flow chart for the decodeHelper method: (a) Java, (b) C++, (c) Python

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

Now let us assume that the value of key = –2 and the character to decode = *a*. Thus, *a* is a supported character that can be decoded. Now the position of *a* = 0. Then we check the position + key = 0 – 2 = –2 < 0. So we wrap around by adding 26 to the sum = –2 + 26 = 24. Thus, we shift by moving 24 characters ahead of *a*. Therefore, *a* is decoded as *y*.

IV. CONCLUSION

Cryptography deals with various techniques that can be used to convey information in a secure fashion. Its objective is to enable the recipient of the message to receive it properly and stop eavesdroppers from understanding it. Cryptography is the art and science of turning an original message into a form that is completely unreadable. The two techniques used to convert data into an unreadable form are the transposition technique and the substitution technique. The Caesar cipher uses the substitution method.

The Caesar cipher algorithm is one of the oldest algorithms. Far newer algorithms that are far more secure have emerged, but the Caesar cipher algorithm is still the fastest because of its simplicity. But it is very easy to crack.

We implemented three programs in this paper based on Java, C++, and Python to implement the Caesar cipher encryption algorithm to aid information security students and help them understand this algorithm. Therefore, it is possible to use this paper for educational purposes in the field of information and communication security. It is crucial to highlight that a code flow chart is used for each program, and a chart is also used to describe the flow of the code. This further reveals the step sequence for major methods used in the code and the relationships between them. Some detailed technical descriptions are also presented for each method used to encode and decode messages. Furthermore, the weaknesses and limitations of the classical Caesar cipher are clearly described.

APPENDIX A. JAVA CODE

```
import java.io.*;
import java.util.*;

public class EncDec {
            static BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
            static PrintWriter write;
            static BufferedReader fileRead;
            static String inFile, outFile;
            static int key;
            static StringTokenizer tok;

            public static void main(String[] args) throws IOException {
                        System.out.print("Welcome to File Encryption/Decryption Program");
                        System.out.println("*****************************************");
                        System.out.println();
                        System.out.print("Enter Source File Name> ");
                        inFile = stdin.readLine();
                        System.out.print("Enter Target File Name> ");
                        outFile = stdin.readLine();
                        System.out.print("Enter Key (>0: encryption, < 0 decryption, 0: copying)> ");
                        key = Integer.parseInt(stdin.readLine());

                        if (key > 0) {
                                    encode();
                        } else if (key < 0) {
                                    decode();
                        } else {
                                    copy();
                        }
                        System.out.println(inFile + " " + outFile + " " + key);
                        System.out.println(".........................");
                        System.out.println("Done, please check output file");
                        write.close();

            }

            public static void encode() throws IOException {
                        fileRead = new BufferedReader(new FileReader(inFile));
                        write = new PrintWriter(new FileWriter(outFile));
                        String line = fileRead.readLine();
                        while (line != null) {
                                    tok = new StringTokenizer(line);
                                    while (tok.hasMoreTokens()) {
                                                String wordIn = tok.nextToken();
                                                String wordOut = "";
                                                for (int i = 0; i < wordIn.length(); i++) {
            char c = wordIn.charAt(i);
            char o = encodeHelper(c);
                                                            wordOut = wordOut + o;
                                                }
                                                write.print(wordOut + " ");
                                    }
                                    write.println();
                                    line = fileRead.readLine();
                        }
            }

            public static void decode() throws IOException {
                        fileRead = new BufferedReader(new FileReader(inFile));
                        write = new PrintWriter(new FileWriter(outFile));
                        String line = fileRead.readLine();
                        while (line != null) {
                                    tok = new StringTokenizer(line);
                                    while (tok.hasMoreTokens()) {
                                                String wordIn = tok.nextToken();
                                                String wordOut = "";
            for (int i = 0; i < wordIn.length(); i++) {
            char c = wordIn.charAt(i);
            char o = decodeHelper(c);
                                                            wordOut = wordOut + o;
                                                }
                                                write.print(wordOut + " ");
                                    }
                                    write.println();
                                    line = fileRead.readLine();
                        }
            }
```


https://sites.google.com/site/ijcsis/
ISSN 1947-5500

```
            public static void copy() throws IOException {
                        fileRead = new BufferedReader(new FileReader(inFile));
                        write = new PrintWriter(new FileWriter(outFile));
                        String line = fileRead.readLine();
                        while (line != null) {
                                    write.println(line);
                                    line = fileRead.readLine();
                        }
            }

            public static char encodeHelper(char temp) {
                        int key = EncDec.key % 26; //added this line to support any key
                        char a = 'a';
                        char A = 'A';
                        char position;
                        if (temp >= a && temp <= 'z')
                                    position = (char) (temp - a);
                        else if (temp >= A && temp <= 'Z')
                                    position = (char) (temp - A);
                        else
                                    return temp;

                        if (position >= (26 - key))
                                    temp = (char) (temp - (26 - key));
                        else
                                    temp = (char) (temp + key);
                        return temp;
            }

            public static char decodeHelper(char temp) {
                        int key = EncDec.key % 26; //added this line to support any key
                        char a = 'a';
                        char A = 'A';

                        //Changed how to calculate position
                        char position;
                        if (temp >= a && temp <= 'z')
                                    position = (char) (temp - a);
                        else if (temp >= A && temp <= 'Z')
                                    position = (char) (temp - A);
                        else
                                    return temp;

                        //changed how to check for out of range
                        if (position + key < 0)
                                    temp = (char) (temp + key + 26);
                        else
                                    temp = (char) (temp + key);
                        return temp;
            }

}
```

APPENDIX B. C++ CODE

```
 // Libraries
#include <string>
#include <vector>
#include <fstream>
#include <iostream>
using namespace std;
static string inFile;
static string outFile;
int key = 0;
// Encode Helper function used in encode function
char encodeHelper(char temp)
{
            // Added this line to support any key
            key = key % 26;
            char a = 'a';
            char A = 'A';
            char position;
            if (temp >= a && temp <= 'z')
            {
                        position = temp - a;
            }
            else if (temp >= A && temp <= 'Z')
            {
                        position = temp - A;
            }
            else
            {
                        return temp;
            }

            if (position >= (26 - key))
            {
                        temp = (temp - (26 - key));
            }
            else
            {
                        temp = (temp + key);
            }
            return temp;
}
// Decode Helper function used in Decode function
char decodeHelper(char temp)
{
            // Added this line to support any key
            key = key % 26;
            char a = 'a';
            char A = 'A';

            // Changed how to calculate position
            char position;
            if (temp >= a && temp <= 'z')
```


https://sites.google.com/site/ijcsis/
ISSN 1947-5500

```
		{
			position = (temp - a);
		}
		else if (temp >= A && temp <= 'Z')
		{
			position = (temp - A);
		}
		else
		{
			return temp;
		}

		// Changed how to check for out of range
		if (position + key < 0)
		{
			temp = (temp + key + 26);
		}
		else
		{
			temp = (temp + key);
		}
		return temp;
}
// Encode function used for Encoding
void encode()
{
		ifstream fileRead (inFile);
		ofstream write (outFile);
		string line;
		// If file exists
		if (fileRead)
		{
			// Read file line by line
			while (getline(fileRead,line))
			{
				// Tokenize line
				char * tok = strtok((char*)(line.c_str()), " ");
				while (tok)
				{
					string wordIn = tok;
					string wordOut = "";
					tok = strtok(NULL, " ");
					// Encode each character of each token using
encodeHelper function
					for (int i = 0; i < wordIn.length(); i++)
					{
						char c = wordIn[i];
						char o = encodeHelper(c);
						wordOut = wordOut + o;
					}
					write << wordOut + " ";
				}
				write<<"\n";
			}
			fileRead.close();
			write.close();
		}
		else
		{
			cout << "Unable to open file"<<endl;
		}
}
// Decode function used to Decode files
void decode()
{
		ifstream fileRead (inFile);
		ofstream write (outFile);
		string line;
		if (fileRead.is_open())
		{
			// Read file to be decoded line by line
			while (getline(fileRead,line))
			{
				// Tokenize line
				char * tok = strtok((char*)(line.c_str()), " ");
				while (tok)
				{
					string wordIn = tok;
					string wordOut = "";
					tok = strtok(NULL, " ");
					// Decode each character of each token
					for (int i = 0; i < wordIn.length(); i++)
					{
						char c = wordIn[i];
						char o = decodeHelper(c);
						wordOut = wordOut + o;
					}
					write << wordOut + " ";
				}
				write<<"\n";
			}
			fileRead.close();
			write.close();
		}
		else
		{
			cout << "Unable to open file"<<endl;
		}
}
// Copy function used for Copying
void copy()
{
		ifstream fileRead (inFile);
		ofstream write (outFile);
		string line;
		if (fileRead.is_open())
		{
			while (getline(fileRead,line))
			{
				write << line <<endl;
			}
			fileRead.close();
			write.close();
		}
		else
		{
```


https://sites.google.com/site/ijcsis/
ISSN 1947-5500

```
                        cout << "Unable to open file"<<endl;
                }
}
// Main Program
void main()
{
                cout << "Welecome to File Encryption/Decryption Program" << endl;
                cout << "**********************************************" << endl;
                cout << endl;
                cout << "Enter Source File Name> ";
                getline (cin,inFile);
                cout << "Enter Target File Name> ";
                getline (cin,outFile);
                cout << "Enter Key (>0: encryption, < 0 decryption, 0: copying)> ";
                cin>>key;

                if (key > 0)
                {
                        encode();
                }
                else if (key < 0)
                {
                        decode();
                }
                else
                {
                        copy();
                }
                cout << inFile << " " << outFile << " " << key << endl;
                cout << "........................." << endl;
                cout << "Done, please check output file" << endl;
}
```

APPENDIX C. PYTHON CODE

```
global inFile
global outFile
global key

def encode():
    fileRead  = open(inFile, "r")
    write = open(outFile, "w")
    for line in fileRead:
        for i in range(0,len(line)):
            write.write(encodeHelper(line[i]))
    write.close()
    fileRead.close()

def encodeHelper(temp):
    k = int(key) % 26

    if temp >= 'a' and temp <= 'z':
        pos = ord(temp) - ord ('a')
    elif temp >= 'A' and temp <= 'Z':
        pos = ord(temp) - ord ('A')
    else:
        return temp

    if(pos >= (26 - k)):
        temp = chr(ord(temp) - 26 + k)
    else:
        temp = chr(ord(temp) + k)

    return temp

def decode():
    fileRead  = open(inFile, "r")
    write = open(outFile, "w")
    for line in fileRead:
        for i in range(0,len(line)):
            write.write(decodeHelper(line[i]))
    write.close()
    fileRead.close()

def decodeHelper(temp):
    k = int(-key) % 26
    k = -k

    if temp >= 'a' and temp <= 'z':
        pos = ord(temp) - ord ('a')
    elif temp >= 'A' and temp <= 'Z':
        pos = ord(temp) - ord ('A')
    else:
        return temp

    if(pos + k < 0):
        temp = chr(ord(temp) + 26 + k)
    else:
        temp = chr(ord(temp) + k)

    return temp

def copy():
    fileRead  = open(inFile, "r")
    write = open(outFile, "w")
    write.write(fileRead.read())
    write.close()
    fileRead.close()

print "Welcome to File Encryption/Decryption Program"
print "**********************************************\n"
print "Enter Source File Name> "
inFile = 'x' # raw input()
print "Enter Target File Name> "
outFile = 'a' # raw_input()
print "Enter Key (>0: encryption, < 0 decryption, 0: copying)> "
key = 0 # raw_input()

if key > 0:
    encode()
```


https://sites.google.com/site/ijcsis/
ISSN 1947-5500

```
elif key < 0:
    decode()
else:
    copy()

print inFile + " " + outFile + " " + str(key)
print "........................."
print "Done, please check output file"
```

REFERENCES

[1] Eskicioglu, A., & Litwin, L. (**2001**). Cryptography. *IEEE Potentials*, *20*(1), 36-38.

[2] Udo, G. J. (2001). Privacy and security concerns as major barriers for e-commerce: a survey study. *Information Management & Computer Security*, *9*(4), 165-174.

[3] Chellappa, R. K., & Pavlou, P. A. (**2002**). Perceived information security, financial liability and consumer trust in electronic commerce transactions. *Logistics Information Management*, *15*(5/6), 358-368.

[4] Goldreich, O. (**2005**). Foundations of Cryptography–A Primer. *Foundations and Trends® in Theoretical Computer Science*, *1*(1), 1-116.

[5] Kahate, A. **(2013)**. *Cryptography and network security*. Tata McGraw-Hill Education.

[6] Calabrese, T. **(2004)**. *Information security intelligence: Cryptographic principles and applications*. Cengage Learning.

[7] Lubbe, J. C. **(1998)**. *Basic methods of cryptography*. Cambridge University Press.

[8] Stallings, W. (**2006**). Cryptography and network security: principles and practices. Pearson Education India.

[9] Ahmad, M., Khan, I. R., & Alam, S. (**2015**). Cryptanalysis of image encryption algorithm based on fractional-order lorenz-like chaotic system. In *Emerging ICT for Bridging the Future-Proceedings of the 49th Annual Convention of the Computer Society of India CSI Volume 2* (pp. 381-388). Springer International Publishing.

[10] Bishop, D. (**2003**). Introduction to cryptography with Java applets. Jones & Bartlett Learning.

[11] Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (**1996**). *Handbook of applied cryptography*. CRC press.

[12] Dooley, J. F. (2013). *A brief history of cryptology and cryptographic algorithms*. Springer.

[13] Manasrah, A. M., & Al-Din, B. N. (**2016**). Mapping private keys into one public key using binary matrices and masonic cipher: Caesar cipher as a case study. *Security and Communication Networks*.

[14] Mishra, A. (**2013**). Enhancing security of caesar cipher using different methods. *International Journal of Research in Engineering and Technology*, *2*(09), 327-332.

[15] Gowda, S. N. (**2016**). Innovative enhancement of the Caesar cipher algorithm for cryptography. In Advances in Computing, Communication, & Automation (ICACCA)(Fall), International Conference on (pp. 1-4). IEEE.

[16] Robling Denning, D. E. (**1982**). *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc.

[17] Knudsen, L. R. (**1998**). Block Ciphers—a survey. In State of the Art in Applied Cryptography (pp. 18-48). Springer Berlin Heidelberg.

[18] Salomon, D. (**2003**). Introduction. In *Data Privacy and Security* (pp. 1-17). Springer New York.

[19] Omolara, O. E., Oludare, A. I., & Abdulahi, S. E. (**2014**). Developing a modified Hybrid Caesar cipher and Vigenere cipher for secure Data Communication. *Computer Engineering and Intelligent Systems*, *5*, 34-46.

[20] Govinda, K. (**2011**). Multilevel cryptography technique using graceful codes. *Journal of Global Research in Computer Science*, *2*(7), 1-5.

[21] Bruen, A. A., & Forcinito, M. A. (**2011**). Cryptography, information theory, and error-correction: a handbook for the 21st century (Vol. 68). John Wiley & Sons.

[22] Jain, A., Dedhia, R., & Patil, A. (**2015**). Enhancing the security of caesar cipher substitution method using a randomized approach for more secure communication. *arXiv preprint arXiv:1512.05483*.

[23] Li, L., Lu, R., Choo, K. K. R., Datta, A., & Shao, J. (**2016**). Privacy-preserving-outsourced association rule mining on vertically partitioned databases. IEEE Transactions on Information Forensics and Security, 11(8), 1847-1861.

AUTHOR PROFILE

Ismail Keshta is an assistant professor at the Department of Computer and Information Technology at Dammam Community College (DCC), King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia. He received the B.Sc. and M.Sc. degrees in computer engineering and the Ph.D. degree in computer science and engineering from the King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia, in 2009, 2011, and 2016, respectively. He was a Lecturer with the Computer Engineering Department, KFUPM, from 2012 to 2016. Prior to that, in 2011, he was a Lecturer with Princess Nora Bint Abdulrahman University (PNU) and Imam Muhammad ibn Saud Islamic University (IMAMU), Riyadh, Saudi Arabia. He His research interests include software process improvement, modeling, and intelligent systems.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500